

**Winter 2006-2007  
Compiler Construction  
T2 – Lexical Analysis  
(Scanning)**

Mooly Sagiv and Roman Manevich  
School of Computer Science  
Tel-Aviv University

---

---

---

---

---

---

---

---

**Material taught in lecture**

- Scanner specification language: regular expressions
- Scanner generation using automata theory + extra book-keeping
- Basic Java tutorial

2

---

---

---

---

---

---

---

---

**Today**

<small>ic</small> IC Language	<small>Lexical Analysis</small>	<small>Syntax Analysis Parsing</small>	<small>AST</small>	<small>Symbol Table etc.</small>	<small>Inter. Rep. (IR)</small>	<small>Code Generation</small>	<small>exe</small> Executable code
----------------------------------	---------------------------------	--	--------------------	----------------------------------	---------------------------------	--------------------------------	---------------------------------------

- Goals:
  - Quick review of lexical analysis theory
  - Implementing a scanner for IC via JFlex
  - Explain PA1

3

---

---

---

---

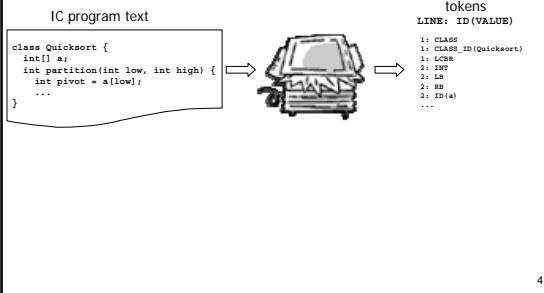
---

---

---

---

## Scanning IC programs




---

---

---

---

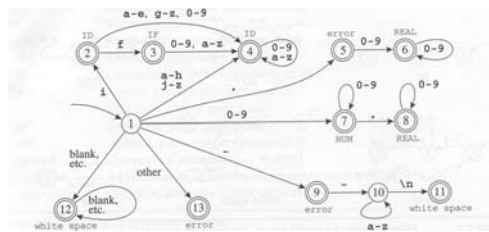
---

---

---

---

## Scanner implementation



What are the outputs on the following inputs:  
**ifelse**  
**if a**  
**.75**  
**89**  
**89.94**

5

---

---

---

---

---

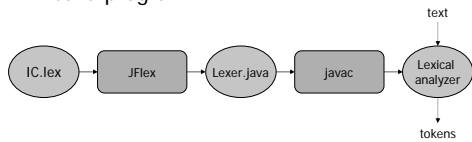
---

---

---

## Lexical analysis with JFlex

- JFlex – fast lexical analyzer generator
  - Recognizes lexical patterns in text
  - Breaks input character stream into tokens
- Input: scanner specification file
- Output: a lexical analyzer (scanner)
  - A Java program



6

---

---

---

---

---

---

---

---

## JFlex spec. file

### User code

- Copied directly to Java file

%%

### JFlex directives

- Define macros, state names

%%

### Lexical analysis rules

- Optional state, regular expression, action
- How to break input to tokens
- Action when token matched

Possible source  
of javac errors  
down the road

DIGIT= [0-9]  
LETTER= [a-zA-Z]  
YYINITIAL

{LETTER}  
{LETTER}{DIGIT}\*

7

---

---

---

---

---

---

---

---

## User code

```
package IC.Lexer;  
import IC.Parser.Symbol;
```

```
...  
any scanner-helper Java code  
...
```

8

---

---

---

---

---

---

---

---

## JFlex directives

### ▪ Directives - control JFlex internals

- %line switches line counting on
- %char switches character counting on
- %class class-name
- %cup CUP compatibility mode
- %type token-class-name
- %public Makes generated class public (package by default)
- %function read-token-method
- %scanerror exception-type-name

### ▪ State definitions

- %state state-name

### ▪ Macro definitions

- macro-name = regex

9

---

---

---

---

---

---

---

---

## Regular expressions

<code>r\$</code>	match reg. exp. <i>r</i> at end of a line
<code>.</code> (dot)	any character except the newline
<code>"..."</code>	verbatim string
<code>{name}</code>	macro expansion
<code>*</code>	zero or more repetitions
<code>+</code>	one or more repetitions
<code>?</code>	zero or one repetitions
<code>(...)</code>	grouping within regular expressions
<code>a b</code>	match <i>a</i> or <i>b</i>
<code>[...]</code>	class of characters - any <u>one</u> character enclosed in brackets
<code>a-b</code>	range of characters
<code>[^...]</code>	negated class - any one not enclosed in brackets

10

---

---

---

---

---

---

---

---

## Example macros

```
ALPHA= [A-Za-z_]
DIGIT= [0-9]
ALPHA_NUMERIC={ALPHA} | {DIGIT}
IDENT={ALPHA} ({ALPHA_NUMERIC}) *
NUMBER= ({DIGIT}) +
WHITE_SPACE= ([\ \n\r\t\f]) +
```

11

---

---

---

---

---

---

---

---

## Lexical analysis rules

- Rule structure
  - `[states] regexp {action as Java code}`
  - regexp pattern - how to break input into tokens
  - Action invoked when pattern matched
  - Priority for rule matching longest string
- More than one match for same length – priority for rule appearing first!
- Important: rules given in a JFlex specification should match all possible inputs!

12

---

---

---

---

---

---

---

---

## Action body

- Java code
- Can use special methods and vars
  - `yytext()`
  - `yyline` (when enabled)
  - ...
- Scanner state transition
  - `yybegin(state-name)`
  - `YYINITIAL`

13

---

---

---

---

---

---

---

---

## More on scanner states

- Independent mini-scanner inside scanner
  - comments `/* */`
  - quote string `" "`
- Scan syntactically different portion of the input
- Tokenize according to context

```
// this conditon checks if x > y  
if (x>y) {  
}
```

- Example
  - "if" is a keyword token when in program text
  - "if" is part of comment text when inside a comment

14

---

---

---

---

---

---

---

---

```
<YYINITIAL> {NUMBER} {  
    return new Symbol(sym.NUMBER, yytext(), yyline);  
}  
<YYINITIAL> {WHITE_SPACE} { }  
  
<YYINITIAL> "+" {  
    return new Symbol(sym.PLUS, yytext(), yyline);  
}  
<YYINITIAL> "-" {  
    return new Symbol(sym.MINUS, yytext(), yyline);  
}  
<YYINITIAL> "*" {  
    return new Symbol(sym.TIMES, yytext(), yyline);  
}  
...  
  
<YYINITIAL> "/*" { yybegin(COMMENTS); }  
<COMMENTS> ["\n"] { }  
<COMMENTS> ["\n"] { yybegin(YYINITIAL); }  
<YYINITIAL> "." { return new Symbol(sym.error, null); }
```

15

---

---

---

---

---

---

---

---

## Putting it all together – count number of lines

```
lineCount.lex
import java_cup.runtime.Symbol;
%%
%cup
%{
    private int lineCounter = 0;
%}

%eofval{
    System.out.println("line number=" + lineCounter);
    return new Symbol(sym.EOF);
%eofval}

NEWLINE=\n
%%
<YYINITIAL>{NEWLINE} {
    lineCounter++;
}
<YYINITIAL>[!{NEWLINE}] { }
```

16

---

---

---

---

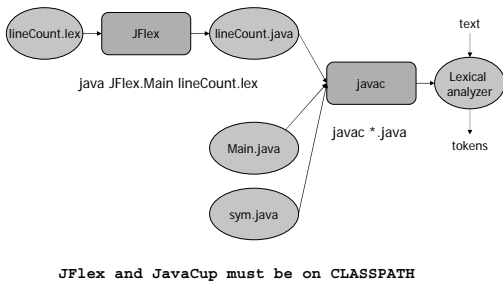
---

---

---

---

## Putting it all together – count number of lines



17

---

---

---

---

---

---

---

---

## Running the scanner

```
import java.io.*;

public class Main {
    public static void main(String[] args) {
        Symbol currToken;
        try {
            FileReader txtFile = new FileReader(args[0]);
            Yylex scanner = new Yylex(txtFile);
            do {
                currToken = scanner.next_token();
                // do something with currToken
            } while (currToken.sym != sym.EOF);
        } catch (Exception e) {
            throw new RuntimeException("IO Error (brutal exit)" +
                e.toString());
        }
    }
}
```

(Just for testing scanner as stand-alone program)

18

---

---

---

---

---

---

---

---

## Common pitfalls

- Classpath
- Path to executable
- Define environment variables
  - JAVA\_HOME
  - CLASSPATH
    - May want to include current directory '.'
- Note the use of . (dot) as part of package name / directory structure
  - e.g., JFlex.Main

19

---

---

---

---

---

---

---

---

## Programming assignment 1

- Implement a scanner for IC
- **class Token**
  - At least – line, id, value
  - Should extend `java_cup.runtime.Symbol`
  - Numeric token ids in `sym.java`
    - Will be later generated by JavaCup
- **class Compiler**
  - Testbed - calls scanner to print list of tokens
- **class LexicalError**
  - Caught by Compiler
- Don't forget to generate scanner **and** recompile Java sources when you change the spec.
- You need to download and install **both** JFlex and JavaCup

20

---

---

---

---

---

---

---

---

## sym.java file

```
public class sym {  
    public static final int EOF = 0;  
    ...  
}
```

- Defines symbol constant ids
- Tells parser what is the token returned by scanner
- Actual value doesn't matter
  - *But different tokens should have different values*
- In the future will be generated by JavaCup

21

---

---

---

---

---

---

---

---

## Token class

```
import java_cup.runtime.Symbol;  
  
public class Token extends Symbol {  
    public int getId() {...}  
    public Object getValue() {...}  
    public int getLine() {...}  
    ...  
}
```

22

---

---

---

---

---

---

---

---

## (some) JFlex directives to use

```
%cup           (integrate with cup)  
%line         (count lines)  
%type Token   (pass type Token)  
%class Lexer  (gen. scanner class)
```

23

---

---

---

---

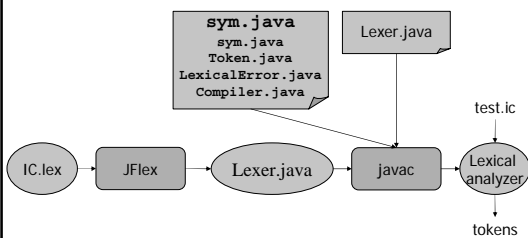
---

---

---

---

## Structure of PA1



24

---

---

---

---

---

---

---

---

## Beginning the assignment

- Download J2SE 1.5
  - Make sure you can compile and run programs
- Download JFlex
- Download JavaCup
- Use of Eclipse is recommended
- Use of Apache Ant is recommended
- JFlex and JavaCup must be in the CLASSPATH
- Use assignment skeleton: [pa1.zip](#) to avoid unnecessary mistakes

25

---

---

---

---

---

---

---

---

## Administrative issues

- Check your details in the list of project teams
  - If you don't have a team, get one by PA deadline
- PA1
  - Electronic submission due Nov. 15, midnight
  - Printout of the README due Nov. 15, noon mailbox  
268 Schreiber
- Carefully READ the material
- Use FORUM

26

---

---

---

---

---

---

---

---

**See you next week**

27

---

---

---

---

---

---

---

---